IN THE UNITED STATES DISTRICT COURT

FOR THE DISTRICT OF DELAWARE

STREAMSERVE AB, Plaintiff, v. EXSTREAM SOFTWARE, LLC and HEWLETT-PACKARD COMPANY, Defendants.	)	C.A. No. _____ **JURY TRIAL DEMANDED**

COMPLAINT AND DEMAND FOR JURY TRIAL

Plaintiff StreamServe AB ("StreamServe"), by and through the undersigned attorneys, hereby files this complaint for patent infringement against Defendants Exstream Software LLC ("Exstream") and Hewlett-Packard Company ("HP"), requesting damages and injunctive relief based upon its personal knowledge as to its own facts and circumstances, and based upon information and belief as to the acts and circumstances of others.

PARTIES

1. StreamServe is a Swedish corporation having its principal place of business at Karlavägen 108, SE-10451 Stockholm, Sweden.

2. Upon information and belief, Exstream is a Delaware limited liability company having its principal place of business at 810 Bull Lea Run, Lexington, Kentucky 40511. Exstream may be served with process by serving its registered agent, Corporation Service Company, 2711 Centerville Road Suite 400, Wilmington, Delaware 19808. Exstream manufactures for sale and/or sells in the United States, and in the District of Delaware, software for enterprise document automation.

3. Upon information and belief, HP is a Delaware corporation having its principal place of business at 3000 Hanover Street, Palo Alto, California 94304. HP may be served with process by serving its registered agent, The Corporation Trust Company, Corporation Trust Center, 1209 Orange Street, Wilmington, Delaware 19801. HP manufactures for sale and/or sells in the United States, and in the District of Delaware, software for enterprise document automation.

JURISDICTION AND VENUE

4. This is an action for patent infringement under the patent laws of the United States, Title 35 of the United States Code, including (but not limited to) 35 U.S.C. § 271.

5. This Court has subject matter jurisdiction pursuant to 28 U.S.C. §§ 1331 and 1338(a).

6. This Court has personal jurisdiction over Exstream. Exstream is incorporated in Delaware and has conducted and continues to conduct business within this judicial district. Upon information and belief, Exstream has directly infringed, contributed to the infringement of, and/or actively induced infringement of, StreamServe's patent within this judicial district.

7. This Court has personal jurisdiction over HP. HP is incorporated in Delaware and has conducted and continues to conduct business within this judicial district. Upon information and belief, HP has directly infringed, contributed to the infringement of, and/or actively induced infringement of, StreamServe's patent within this judicial district.

8. Venue is proper in this Court under 28 U.S.C §§ 1391 and 1400(b) because Exstream and HP have committed acts of infringement in and are subject to personal jurisdiction in this judicial district.

COUNT 1: PATENT INFRINGEMENT BY EXSTREAM

9. StreamServe incorporates paragraphs 1-8 as if fully set forth herein.

10. United States Patent No. 7,127,520 (the "'520 Patent"), entitled "Method and System for Transforming Input Data Streams," was duly and legally issued by the United States Patent and Trademark Office on October 24, 2006. A copy of the '520 Patent is attached hereto as Exhibit A.

11. StreamServe is the owner by valid assignment of the entire right, title, and interest in and to the '520 Patent and possesses all rights of recovery under the '520 Patent, including the right to recover damages for past infringement.

12. Upon information and belief, Exstream has been and is now making, using, selling, and offering for sale within the United States, or importing into the United States, products, including the Exstream Software by HP Dialogue software solution, that infringe and can be used to infringe claims of the '520 Patent.

13. Exstream has been contributing to the infringement of and/or actively inducing the infringement of the '520 Patent by others by, among other things, distributing or offering for sale products, including the Exstream Software by HP Dialogue software solution, and documentation that teaches third parties to use said products in a manner that directly infringes claims of the '520 Patent.

14. StreamServe has no adequate remedy at law against Exstream's acts of infringement and StreamServe will suffer irreparable harm unless Exstream is preliminarily and permanently enjoined from its infringement of the '520 Patent.

15. Exstream has had knowledge of the '520 Patent since at least October 30, 2006 and has not ceased its infringing activity. Exstream's infringement of the '520 Patent has been and continues to be willful and deliberate.

16. Exstream, by way of its infringing activity, has caused and continues to cause StreamServe to suffer damages in an amount to be determined at trial.

COUNT 2: PATENT INFRINGEMENT BY HP

17. StreamServe incorporates paragraphs 1-16 as if fully set forth herein.

18. Upon information and belief, HP has been and is now making, using, selling, and offering for sale within the United States, or importing into the United States, products, including the Exstream Software by HP Dialogue software solution, that infringe and can be used to infringe claims of the '520 Patent.

19. HP has been and now is contributing to the infringement of and/or actively inducing the infringement of the '520 Patent by other by, among other things, distributing or offering for sale products, including the Exstream Software by HP Dialogue software solution, and documentation that teaches third parties to use said products in a manner that directly infringes claims of the '520 Patent.

20. StreamServe has no adequate remedy at law against HP's acts of infringement and StreamServe will suffer irreparable harm unless HP is preliminarily and permanently enjoined from its infringement of the '520 Patent.

21. HP has had knowledge of the '520 Patent since at least January 22, 2008 and has not ceased its infringing activity. HP's infringement of the '520 Patent has been and continues to be willful and deliberate.

22. HP, by way of its infringing activity, has caused and continues to cause StreamServe to suffer damages in an amount to be determined at trial.

PRAYER FOR RELIEF

WHEREFORE, StreamServe prays for the following relief:

A. A judgment in favor of StreamServe that Exstream has infringed, directly and indirectly by way of inducement and/or contributory infringement, literally and/or under the doctrine of equivalents, claims of the '520 Patent;

B. A judgment in favor of StreamServe that HP has infringed, directly and indirectly by way of inducement and/or contributory infringement, literally and/or under the doctrine of equivalents, claims of the '520 Patent;

C. A permanent injunction, enjoining Exstream and its officers, directors, agents, servants, employees, affiliates, divisions, branches, subsidiaries, parents and all others acting in concert or privity with any of them from infringing, inducing the infringement of, or contributing to the infringement of the '520 Patent;

D. A permanent injunction, enjoining HP and its officers, directors, agents, servants, employees, affiliates, divisions, branches, subsidiaries, parents and all others acting in concert or privity with any of them from infringing, inducing the infringement of, or contributing to the infringement of the '520 Patent;

E. An award to StreamServe of the damages to which it is entitled under 35 U.S.C. § 284 for Exstream's past infringement and any continuing or future infringement up until the date Exstream is finally and permanently enjoined from further infringement, including both compensatory damages and treble damages for willful infringement;

F. An award to StreamServe of the damages to which it is entitled under 35 U.S.C. § 284 for HP's past infringement and any continuing or future infringement up until the date HP is finally and permanently enjoined from further infringement, including both compensatory damages and treble damages for willful infringement;

G. A declaration that this is an "exceptional" case within the meaning of 35 U.S.C. § 285, entitling StreamServe to an award of its attorney's fees, expenses, and costs;

H. A judgment and order requiring Exstream to pay StreamServe the costs incurred in bringing this action (including all disbursements), as well as attorney's fees as provided by 35 U.S.C. § 285;

I. A judgment and order requiring HP to pay StreamServe the costs incurred in bringing this action (including all disbursements), as well as attorney's fees as provided by 35 U.S.C. § 285;

J. An award to StreamServe of pre-judgment and post-judgment interest on damages; and

K. Such other and further relief in law or in equity to which StreamServe may be justly entitled.

JURY TRIAL DEMAND

StreamServe respectfully demands a trial by jury of any and all issues triable of right before a jury.

John W. Shaw (No. 3362)
Monté T. Squire (No. 4764)
Young Conaway Stargatt & Taylor, LLP
The Brandywine Building
1000 West Street, 17th Floor
Wilmington, Delaware 19801
(302) 571-6600
msquire@ycst.com

Of Counsel:
Robert M. Isackson
Rodger A. Sadler
Joseph A. Sherinsky
Orrick, Herrington & Sutcliffe LLP
666 Fifth Avenue
New York, New York 10103
(212) 506-5000

Dated: June 9, 2008

CIVIL COVER SHEET

The JS 44 civil cover sheet and the information contained herein neither replace nor supplement the filing and service of pleadings or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. (SEE INSTRUCTIONS ON THE REVERSE OF THE FORM.)

I. (a) PLAINTIFFS

STREAMSERVE AB

(b) County Of Residence Of First Listed Plaintiff: (Except In U.S. Plaintiff Cases)

(c) Attorneys (Firm Name, Address, And Telephone Number)
John W. Shaw (No. 3362), Monté T. Squire (No. 4764)
Young Conaway Stargatt & Taylor, LLP, The Brandywine Building,
1000 West Street, 17th Floor, Wilmington, DE 19801

DEFENDANT(S)

EXSTREAM SOFTWARE, LLC, HEWLETT-PACKARD COMPANY.

County Of Residence Of First Listed Defendant: (IN U.S. PLAINTIFF CASES ONLY)
NOTE: IN LAND CONDEMNATION CASES, USE THE LOCATION OF THE TRACT OF LAND INVOLVED

Attorneys (If Known)

II. BASIS OF JURISDICTION (PLACE AN X IN ONE BOX ONLY)

- ☐ 1 U.S. Government Plaintiff
- ☒ 3 Federal Question (U.S. Government Not a Party)
- ☐ 2 U.S. Government Defendant
- ☐ 4 Diversity (Indicate Citizenship of Parties in Item III)

III. CITIZENSHIP OF PRINCIPAL PARTIES (Place An X In One Box For Plaintiff And One Box For Defendant) (For Diversity Cases Only)

	PTF	DEF		PTF	DEF
Citizen of This State	☐ 1	☐ 1	Incorporated or Principal Place of Business in This State	☐ 4	☐ 4
Citizen of Another State	☐ 2	☐ 2	Incorporated and Principal Place of Business in This State	☐ 5	☐ 5
Citizen or Subject of a Foreign Country	☐ 3	☐ 3	Foreign Nation	☐ 6	☐ 6

V. NATURE OF SUIT (Place An X In One Box Only)

CONTRACT

- ☐ 110 Insurance
- ☐ 120 Marine
- ☐ 130 Miller Act
- ☐ 140 Negotiable Instrument
- ☐ 150 Recovery of Overpayment & Enforcement of Judgment
- ☐ 151 Medicare Act
- ☐ 152 Recovery of Defaulted (Excl. Veterans)
- ☐ 153 Recovery of Overpayment of Veteran's Benefits
- ☐ 160 Stockholders' Suits
- ☐ 190 Other Contract
- ☐ 195 Contract Product Liability

REAL PROPERTY

- ☐ 210 Land Condemnation
- ☐ 220 Foreclosure
- ☐ 230 Rent Lease & Ejectment
- ☐ 240 Torts to Land
- ☐ 245 Tort Product Liability
- ☐ 290 All Other Real Property

TORTS

PERSONAL INJURY
- ☐ 310 Airplane
- ☐ 315 Airplane Product Liability
- ☐ 320 Assault, Libel & Slander
- ☐ 330 Federal Employers Liability
- ☐ 340 Marine
- ☐ 345 Marine Product Liability
- ☐ 350 Motor Vehicle
- ☐ 355 Motor Vehicle Product Liability
- ☐ 360 Other Personal Injury

PERSONAL INJURY
- ☐ 362 Personal Injury - Med Malpractice
- ☐ 365 Personal Injury - Product Liability
- ☐ 368 Asbestos Personal Injury Product Liability

PERSONAL ROPERTY
- ☐ 370 Other Fraud
- ☐ 371 Truth in Lending
- ☐ 380 Other Personal Property Damage
- ☐ 385 Property Damage Product Liability

CIVIL RIGHTS

- ☐ 441 Voting
- ☐ 442 Employment
- ☐ 443 Housing/ Accommodations
- ☐ 444 Welfare
- ☐ 440 Other Civil Rights

PRISONER PETITIONS

- ☐ 510 Motions to Vacate Sentence Habeas Corpus
- ☐ 530 General
- ☐ 535 Death Penalty
- ☐ 540 Mandamus & Other
- ☐ 550 Civil Rights
- ☐ 555 Prison Condition

FORFEITURE/PENALTY

- ☐ 610 Agriculture
- ☐ 620 Other Food & Drug
- ☐ 625 Drug Related Seizure of Property 21 U.S.C. 881
- ☐ 630 Liquor Laws
- ☐ 640 R R & Truck
- ☐ 650 Airline Regs
- ☐ 660 Occupational Safety/Health
- ☐ 690 Other

LABOR

- ☐ 710 Fair Labor Standards Act
- ☐ 720 Labor/Mgmt Relations
- ☐ 730 Labor/Mgmt. Reporting & Disclosure Act
- ☐ 740 Railway Labor Act
- ☐ 790 Other Labor Litigation
- ☐ 791 Empl Ret Inc Security Act

BANKRUPTCY

- ☐ 422 Appeal 28 U.S.C. 158
- ☐ 423 Withdrawal 28 U.S.C. 157

PROPERTY RIGHTS

- ☐ 820 Copyrights
- ☒ 830 Patent
- ☐ 840 Trademark

SOCIAL SECURITY

- ☐ 861 HIA (1395ff)
- ☐ 862 Black Lung (923)
- ☐ 863 DIWC/DIWW (405(g))
- ☐ 864 SSID Title XVI
- ☐ 865 RSI (405(g))

FEDERAL TAX SUITS

- ☐ 870 Taxes (U.S. Plaintiff or Defendant)
- ☐ 871 IRS - Third Party 26 U.S.C. 7609

OTHER STATUTES

- ☐ 400 State Reapportionment
- ☐ 410 Antitrust
- ☐ 430 Banks and Banking
- ☐ 450 Commerce/ICC Rates, etc.
- ☐ 460 Deportation
- ☐ 470 Racketeer Influenced and Corrupt Organizations
- ☐ 810 Selective Service
- ☐ 850 Securities/Commodities/ Exchange
- ☐ 875 Customer Challenge 12 U.S.C. 3410
- ☐ 891 Agricultural Acts
- ☐ 892 Economic Stabilization Act
- ☐ 893 Environmental Matters
- ☐ 894 Energy Allocation Act
- ☐ 895 Freedom of Information Act
- ☐ 900 Appeal of Fee Determination Under Equal Access to Justice
- ☐ 950 Constitutionality of State Statutes
- ☐ 890 Other Statutory Actions

IV. ORIGIN (PLACE AN "X" IN ONE BOX ONLY)

☒ 1 Original Proceeding ☐ 2 Removed from Court ☐ 3 Remanded from Appellate Court ☐ 4 Reinstated or Reopened ☐ 5 Transferred from another district (specify) ☐ 6 Multidistrict Litigation ☐ 7 Appeal to District Judge from Magistrate Judgment

VI. CAUSE OF ACTION (CITE THE U.S. CIVIL STATUTE UNDER WHICH YOU ARE FILING AND WRITE BRIEF STATEMENT OF CAUSE DO NOT CITE JURISDICTIONAL STATUTES UNLESS DIVERSITY.):

Brief description of cause:
35 U.S.C. §§ 271 *et seq.*, action for patent infringement

VII. REQUESTED IN COMPLAINT: CHECK IF THIS IS A UNDER F.R.C.P. 23 **CLASS ACTION** ☐ YES ☒ NO **DEMAND $**

Check YES only if demanded in complaint
JURY DEMAND: ☒ YES ☐ NO

VIII. RELATED CASE(S) (See instructions) **IF ANY** JUDGE: DOCKET NUMBER:

6-9-08
DATE SIGNATURE OF ATTORNEY OF RECORD

FOR OFFICE USE ONLY

RECEIPT # _____ AMOUNT _____ APPLYING IFP _____ JUDGE _____ MAG. JUDGE _____

INSTRUCTIONS FOR ATTORNEYS COMPLETING CIVIL COVER SHEET FORM JS-44

Authority For Civil Cover Sheet

The JS-44 civil cover sheet and the information contained herein neither replaces nor supplements the filings and service of pleading or other papers as required by law, except as provided by local rules of court. This form, approved by the Judicial Conference of the United States in September 1974, is required for the use of the Clerk of Court for the purpose of initiating the civil docket sheet. Consequently a civil cover sheet is submitted to the Clerk of Court for each civil complaint filed. The attorney filing a case should complete the form as follows:

I. (a) Plaintiffs - Defendants. Enter names (last, first, middle initial) of plaintiff and defendant. If the plaintiff or defendant is a government agency, use only the full name or standard abbreviations. If the plaintiff or defendant is an official within a government agency, identify first the agency and then the official, giving both name and title.

(b) County of Residence. For each civil case filed, except U.S. plaintiff cases, enter the name of the county where the first listed plaintiff resides at the time of filing. In U.S. plaintiff cases, enter the name of the county in which the first listed defendant resides at the time of filing. (NOTE: In land condemnation cases, the county of residence of the "defendant" is the location of the tract of land involved).

(c) Attorneys. Enter firm name, address, telephone number, and attorney of record. If there are several attorneys, list them on an attachment, noting in this section "(see attachment)."

II. Jurisdiction. The basis of jurisdiction is set forth under Rule 8(a), F.R.C.P., which requires that jurisdictions be shown in pleadings. Place an "X" in one of the boxes. If there is more than one basis of jurisdiction, precedence is given in the order shown below.

United States plaintiff. (1) Jurisdiction is based on 28 U.S.C. 1345 and 1348. Suits by agencies and officers of the United States are included here.

United States defendant. (2) When the plaintiff is suing the United States, its officers or agencies, place an "X" in this box.

Federal question. (3) This refers to suits under 28 U.S.C. 1331, where jurisdiction arises under the Constitution of the United States, an amendment to the Constitution, an act of Congress or a treaty of the United States. In cases where the U.S. is a party, the U.S. plaintiff or defendant code takes precedence, and box 1 or 2 should be marked.

Diversity of citizenship. (4) This refers to suits under 28 U.S.C. 1332, where parties are citizens of different states. When Box 4 is checked, the citizenship of the different parties must be checked. (See Section III below; federal question actions take precedence over diversity cases.)

III. Residence (citizenship) of Principal Parties. This section of the JS-44 is to be completed if diversity of citizenship was indicated above. Mark this section for each principal party.

IV. Cause of Action. Report the civil statute directly related to the cause of action and give a brief description of the cause.

V. Nature of Suit. Place an "X" in the appropriate box. If the nature of suit cannot be determined, be sure the cause of action, in Section IV above, is sufficient to enable the deputy clerk or the statistical clerks in the Administrative Office to determine the nature of suit. If the cause fits more than one nature of suit, select the most definitive.

VI. Origin. Place an "X" in one of the seven boxes.

Original Proceedings. (1) Cases which originate in the United States district courts.

Removed from State Court. (2) Proceedings initiated in state courts may be removed to the district courts under Title 28 U.S.C. Section 1441. When the petition for removal is granted, check this box.

Remanded from Appellate Court. (3) Check this box for cases remanded to the district court for further action. Use the date of remand as the filing date.

Reinstated or Reopened. (4) Check this box for cases reinstated or reopened in the district court. Use the reopening date as the filing date.

Transferred from Another District. (5) For cases transferred under Title 28 U.S.C. Section 1404(a). Do not use this for within district transfers or multidistrict litigation transfers.

Multidistrict Litigation. (6) Check this box when a multidistrict case is transferred into the district under authority of title 28 U.S.C. Section 1407. When this box is checked, do not check (5) above.

Appeal to District Judge from Magistrate Judgment. (7) Check this box for an appeal from a magistrate's decision.

VII. Requested in Complaint. Class Action. Place an "X" in this box if you are filing a class action under Rule 23, F.R.Cv.P.

Demand. In this space enter the dollar amount (in thousands of dollars) being demanded or indicate other demand such as a preliminary injunction.

Jury Demand. Check the appropriate box to indicate whether or not a jury is being demanded.

VIII. Related Cases. This section of the JS-44 is used to reference relating pending cases if any. If there are related pending cases, insert the docket numbers and the corresponding judge names for such cases.

Date and Attorney Signature. Date and sign the civil cover sheet.

US007127520B2

(12) **United States Patent**
Ladd et al.

(10) **Patent No.: US 7,127,520 B2**
(45) **Date of Patent: Oct. 24, 2006**

(54) **METHOD AND SYSTEM FOR TRANSFORMING INPUT DATA STREAMS**

(75) Inventors: **Dennis D. Ladd**, Acton, MA (US); **Anders Hermansson**, Gothenburg (SE)

(73) Assignee: **Streamserve**, Burlington, MA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 629 days.

(21) Appl. No.: **10/184,430**

(22) Filed: **Jun. 28, 2002**

(65) **Prior Publication Data**

US 2004/0024897 A1 Feb. 5, 2004

(51) **Int. Cl.**
G06F 15/16 (2006.01)
G06F 15/80 (2006.01)
H04N 7/12 (2006.01)

(52) **U.S. Cl.** **709/231**; 345/505; 375/240.25; 375/240.26; 709/246

(58) **Field of Classification Search** ..... None
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,275,536 B1 * 8/2001 Chen et al. 375/240.25
6,748,020 B1 * 6/2004 Eifrig et al. 375/240.26
2003/0085902 A1 * 5/2003 Vogelaar et al. 345/505

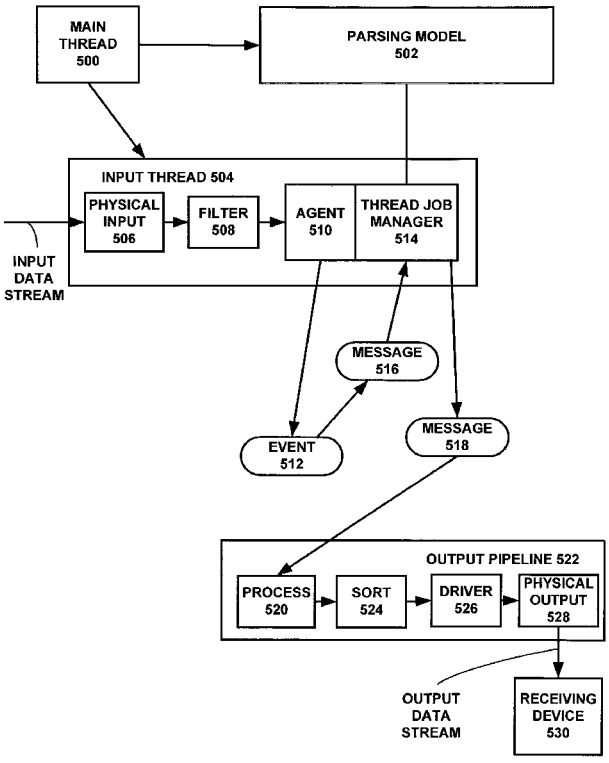
* cited by examiner

Primary Examiner—William Vaughn
Assistant Examiner—Philip Lee
(74) *Attorney, Agent, or Firm*—Goodwin Procter LLP

(57) **ABSTRACT**

The present system and method transforms an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats. A plurality of input connector modules receive respective input data streams and at least one input queue stores the received input data streams. A plurality of job threads is operatively connected to the at least one input queue, each job thread, in parallel with at least one other job thread, formatting a stored input data stream to produce an output data stream. At least one output queue respectively stores the output data streams from the plurality of job threads. A plurality of output connector modules is operatively connected to the at least one output queue, the output connector modules supplying respective output data streams.

27 Claims, 8 Drawing Sheets



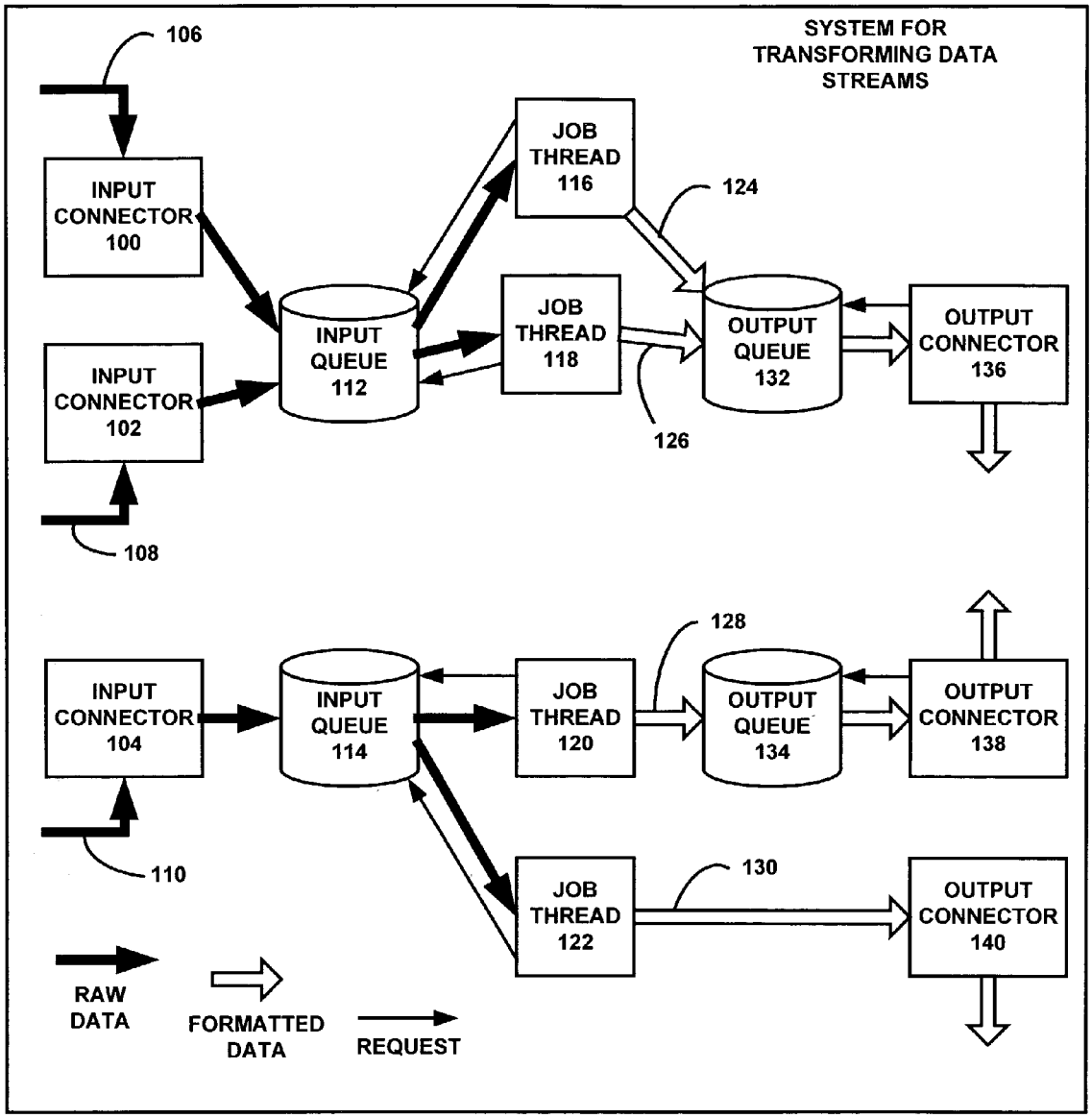

SYSTEM FOR TRANSFORMING DATA STREAMS
106
INPUT CONNECTOR 100
INPUT CONNECTOR 102
108
INPUT QUEUE 112
JOB THREAD 116
JOB THREAD 118
124
126
OUTPUT QUEUE 132
OUTPUT CONNECTOR 136
INPUT CONNECTOR 104
110
INPUT QUEUE 114
JOB THREAD 120
JOB THREAD 122
128
130
OUTPUT QUEUE 134
OUTPUT CONNECTOR 138
OUTPUT CONNECTOR 140
RAW DATA
FORMATTED DATA
REQUEST

FIG. 1

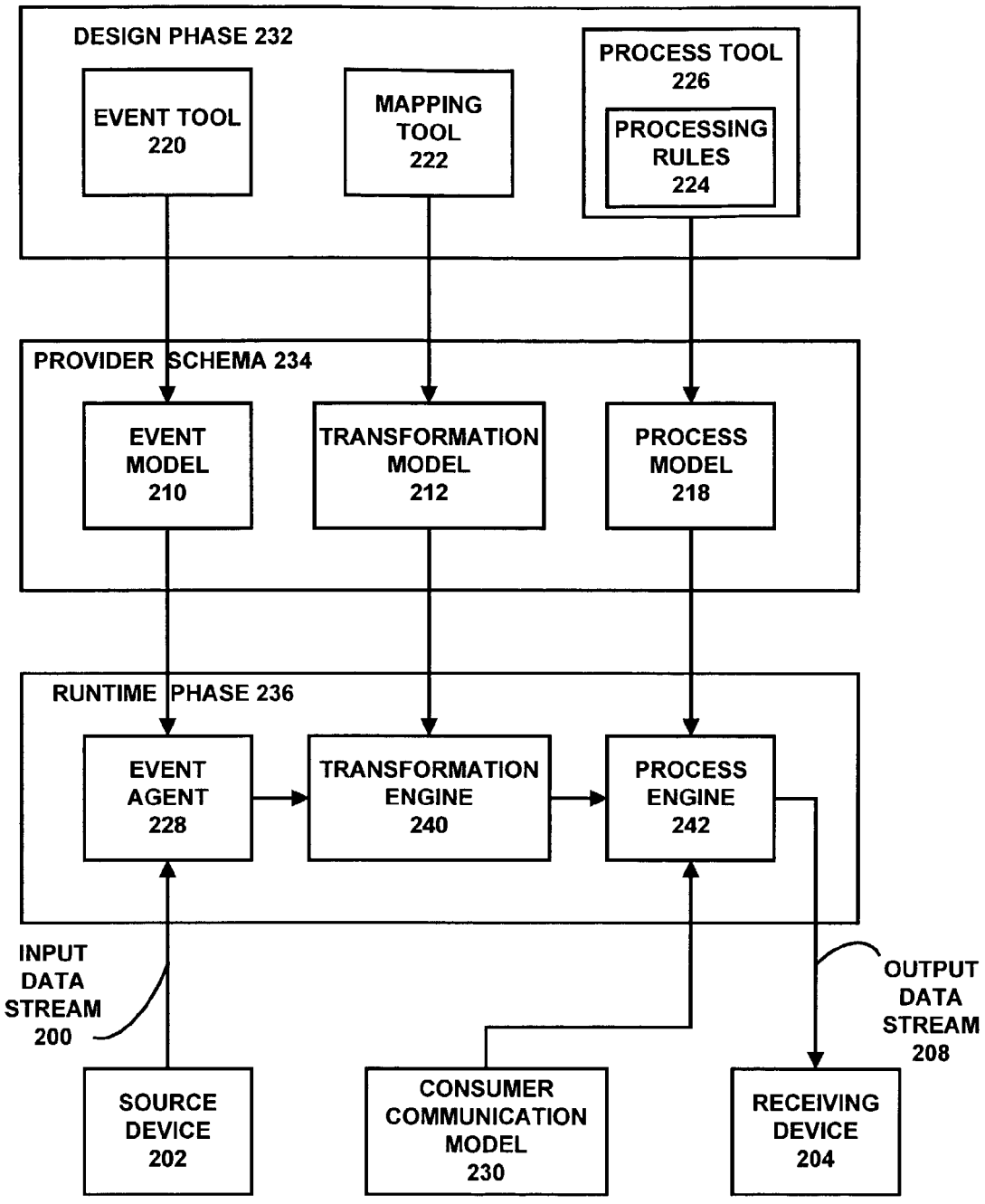

DESIGN PHASE 232
EVENT TOOL 220
MAPPING TOOL 222
PROCESS TOOL 226
PROCESSING RULES 224
PROVIDER SCHEMA 234
EVENT MODEL 210
TRANSFORMATION MODEL 212
PROCESS MODEL 218
RUNTIME PHASE 236
EVENT AGENT 228
TRANSFORMATION ENGINE 240
PROCESS ENGINE 242
INPUT DATA STREAM 200
OUTPUT DATA STREAM 208
SOURCE DEVICE 202
CONSUMER COMMUNICATION MODEL 230
RECEIVING DEVICE 204

FIG. 2

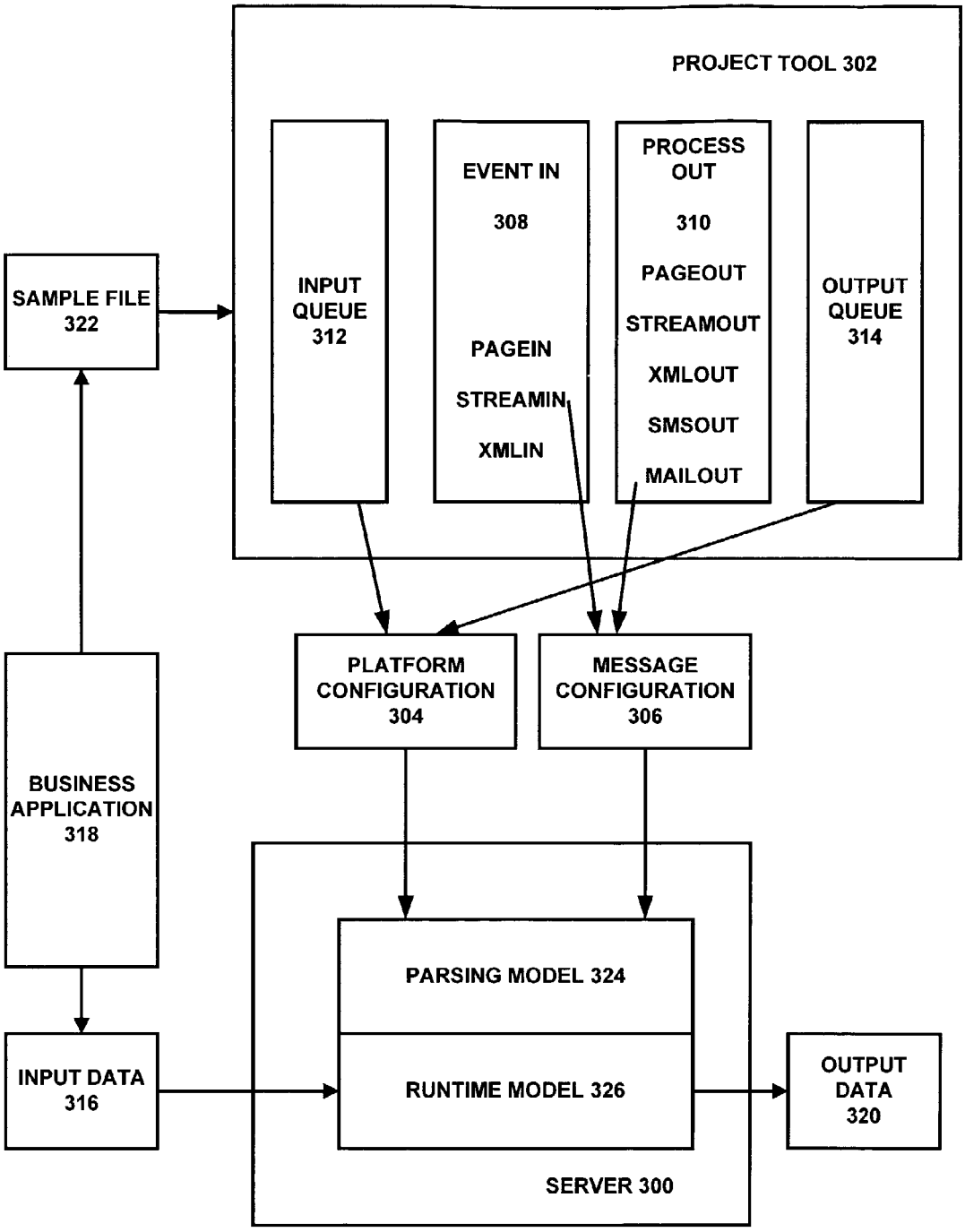

PROJECT TOOL 302
INPUT QUEUE 312
EVENT IN 308
PAGEIN
STREAMIN
XMLIN
PROCESS OUT 310
PAGEOUT
STREAMOUT
XMLOUT
SMSOUT
MAILOUT
OUTPUT QUEUE 314
SAMPLE FILE 322
PLATFORM CONFIGURATION 304
MESSAGE CONFIGURATION 306
BUSINESS APPLICATION 318
PARSING MODEL 324
RUNTIME MODEL 326
INPUT DATA 316
OUTPUT DATA 320
SERVER 300

FIG. 3

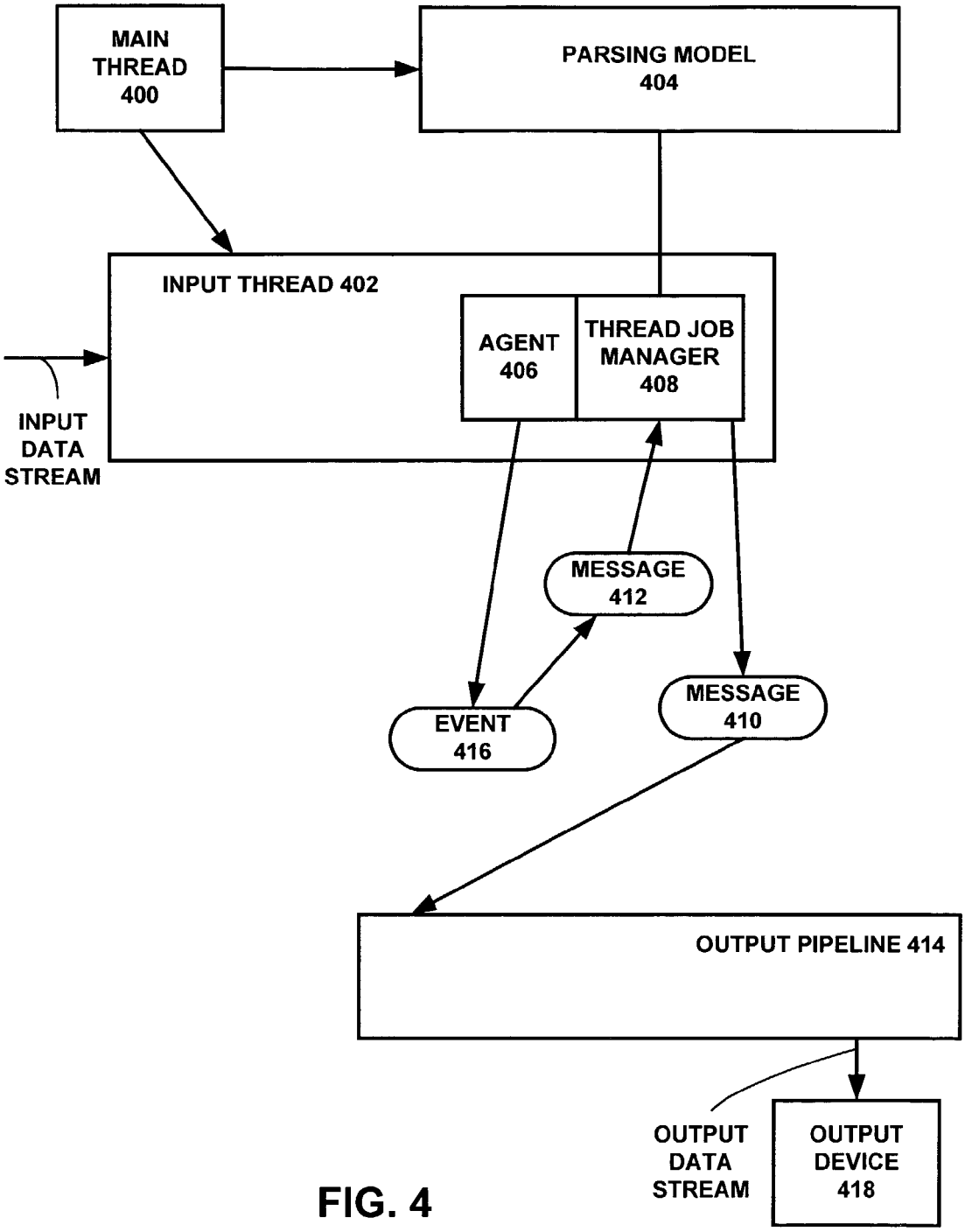

MAIN THREAD 400
PARSING MODEL 404
INPUT THREAD 402
AGENT 406
THREAD JOB MANAGER 408
INPUT DATA STREAM
MESSAGE 412
EVENT 416
MESSAGE 410
OUTPUT PIPELINE 414
OUTPUT DATA STREAM
OUTPUT DEVICE 418

FIG. 4

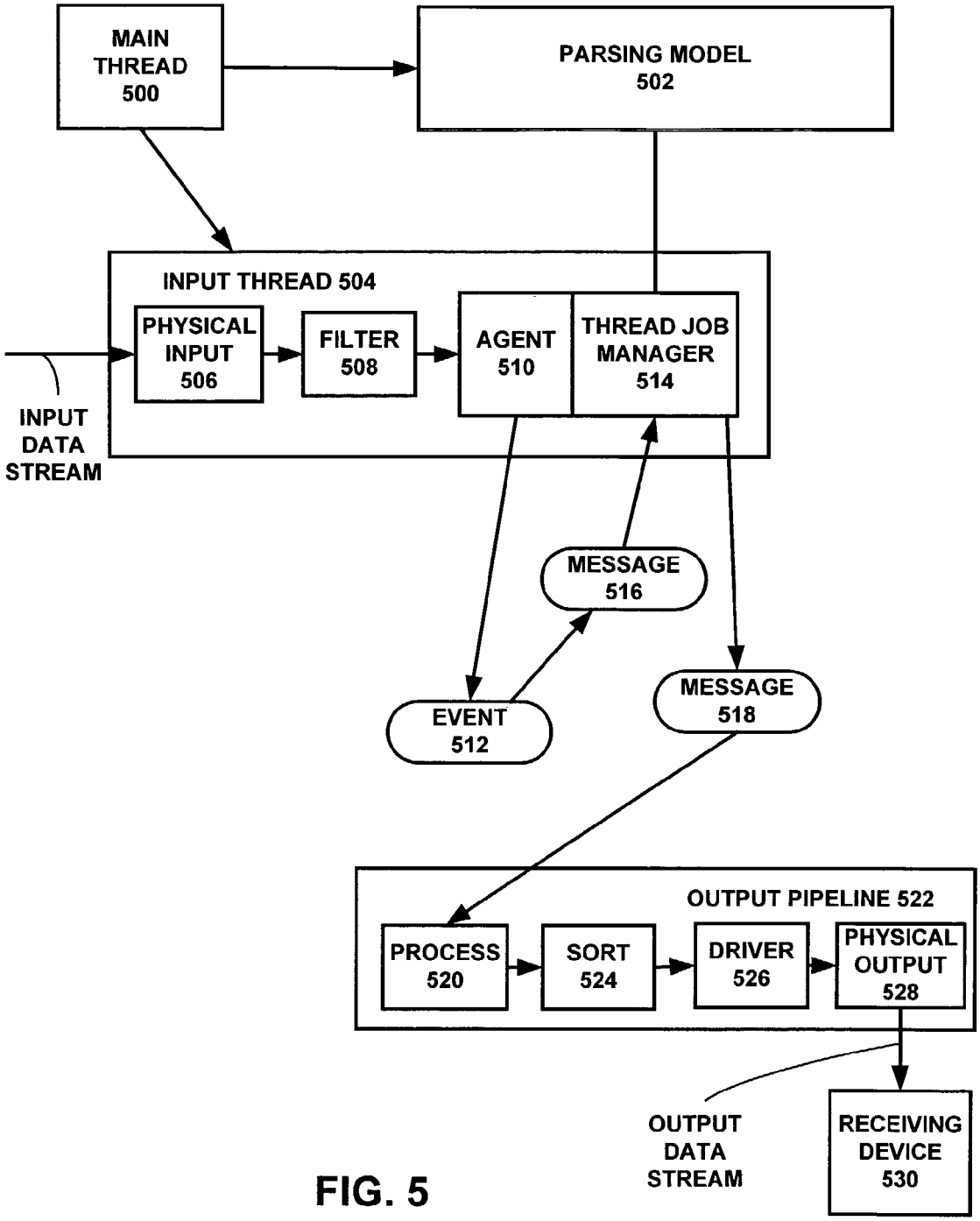

MAIN THREAD 500
PARSING MODEL 502
INPUT THREAD 504
PHYSICAL INPUT 506
FILTER 508
AGENT 510
THREAD JOB MANAGER 514
INPUT DATA STREAM
MESSAGE 516
EVENT 512
MESSAGE 518
OUTPUT PIPELINE 522
PROCESS 520
SORT 524
DRIVER 526
PHYSICAL OUTPUT 528
OUTPUT DATA STREAM
RECEIVING DEVICE 530

FIG. 5

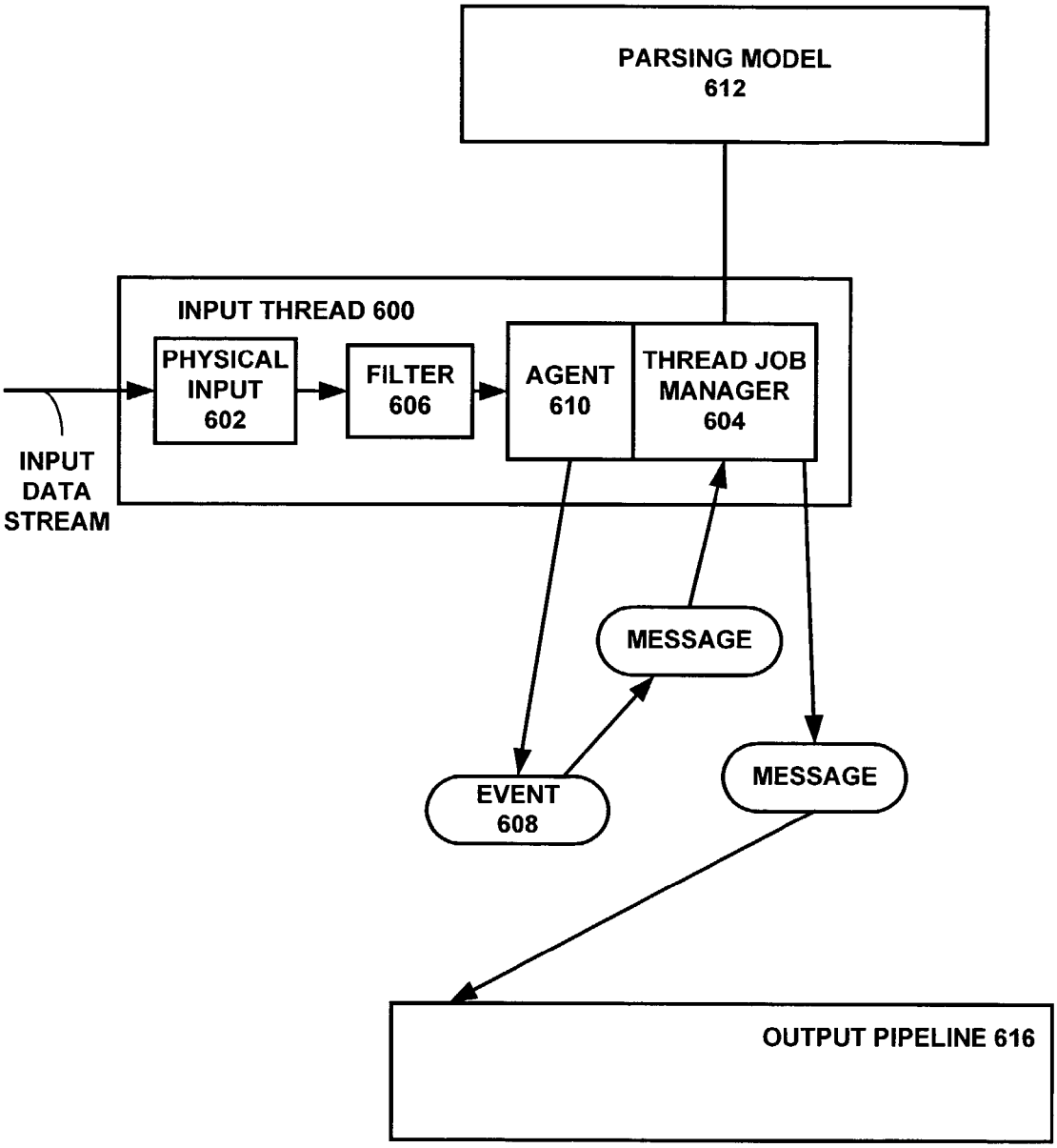

PARSING MODEL
612
INPUT THREAD 600
PHYSICAL INPUT 602
FILTER 606
AGENT 610
THREAD JOB MANAGER 604
INPUT DATA STREAM
MESSAGE
EVENT 608
MESSAGE
OUTPUT PIPELINE 616

FIG. 6

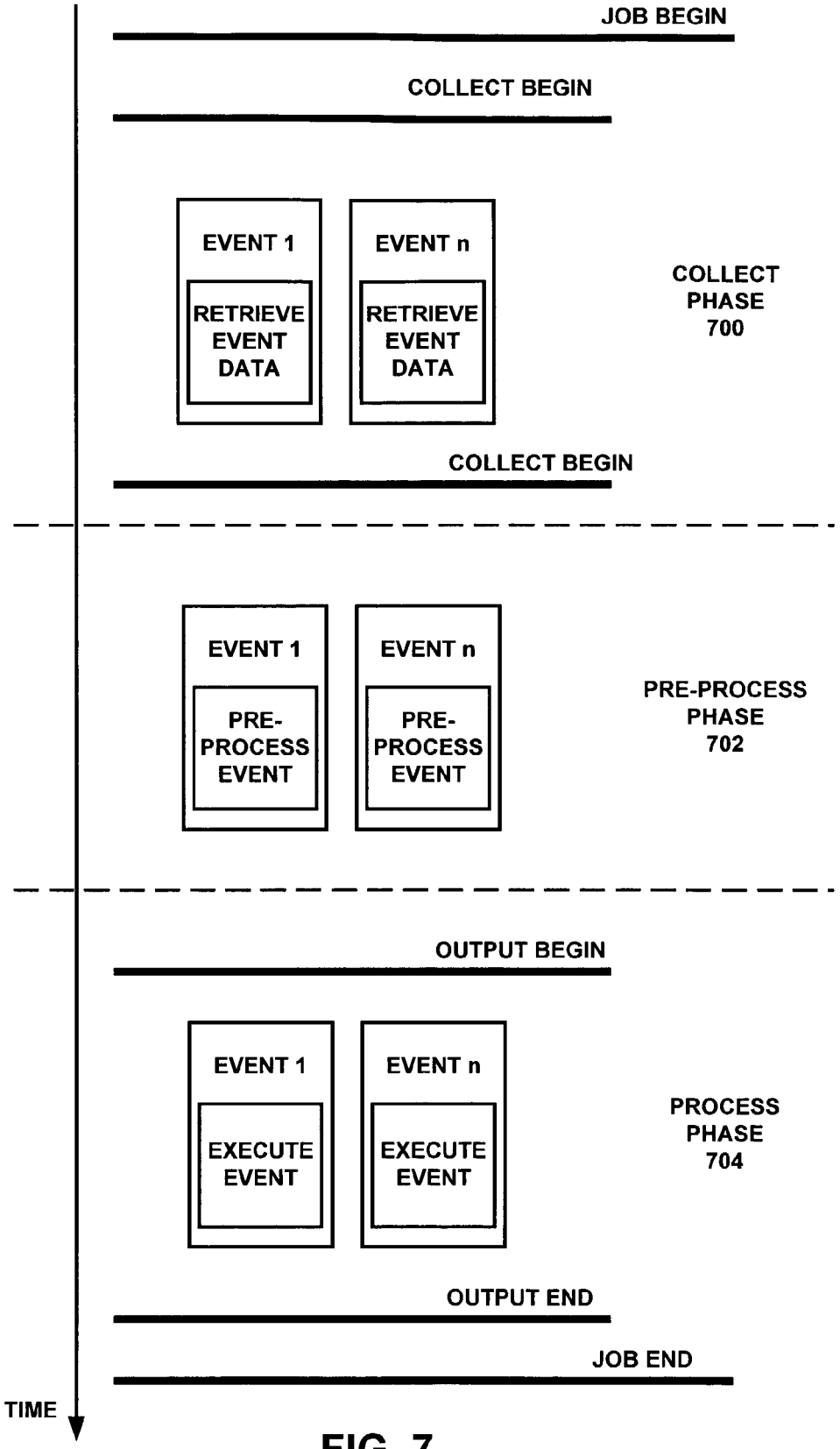

JOB BEGIN
COLLECT BEGIN
EVENT 1
RETRIEVE EVENT DATA
EVENT n
RETRIEVE EVENT DATA
COLLECT PHASE 700
COLLECT BEGIN
EVENT 1
PRE-PROCESS EVENT
EVENT n
PRE-PROCESS EVENT
PRE-PROCESS PHASE 702
OUTPUT BEGIN
EVENT 1
EXECUTE EVENT
EVENT n
EXECUTE EVENT
PROCESS PHASE 704
OUTPUT END
JOB END
TIME

FIG. 7

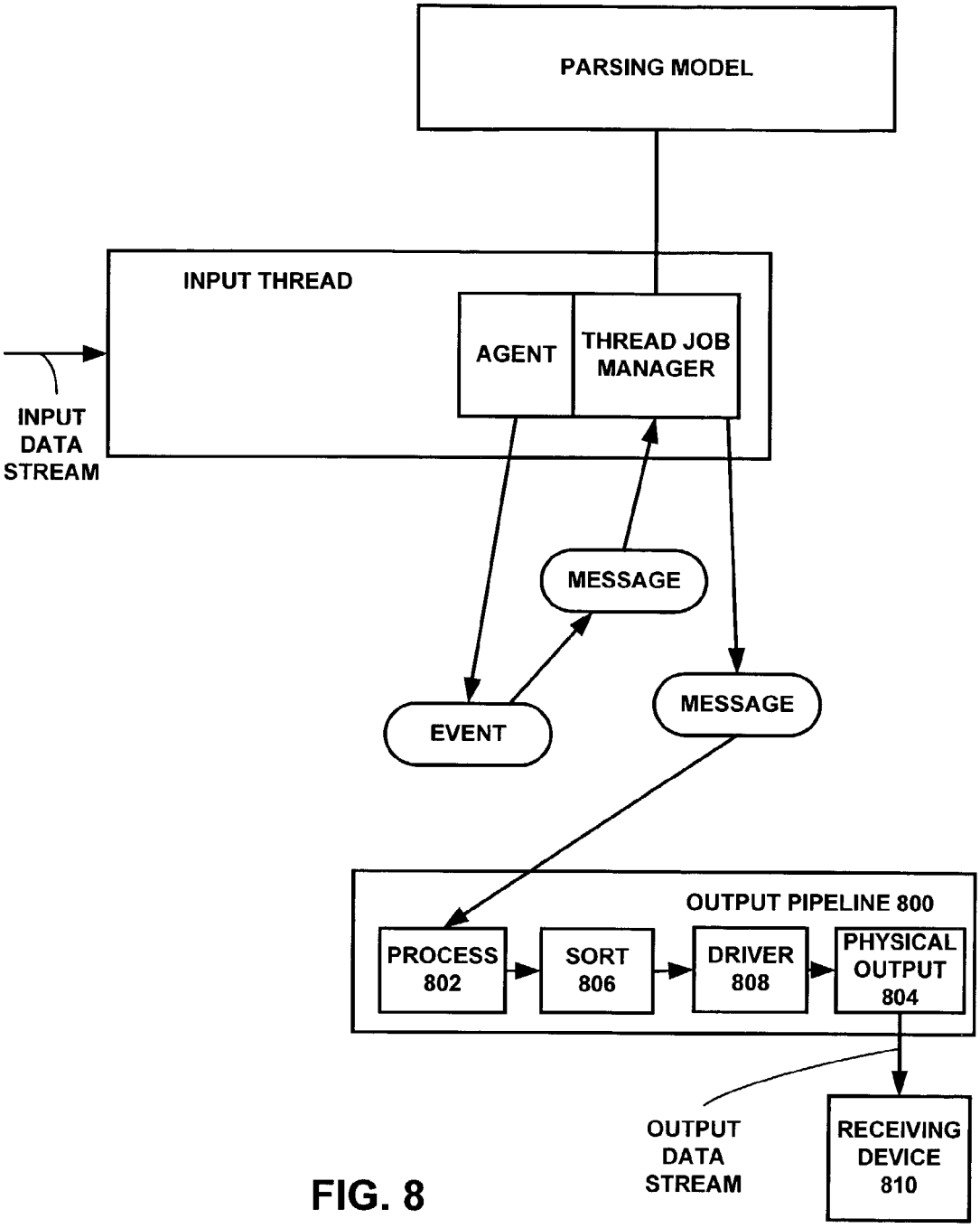

PARSING MODEL
INPUT THREAD
AGENT
THREAD JOB MANAGER
INPUT DATA STREAM
MESSAGE
EVENT
MESSAGE
OUTPUT PIPELINE 800
PROCESS 802
SORT 806
DRIVER 808
PHYSICAL OUTPUT 804
OUTPUT DATA STREAM
RECEIVING DEVICE 810

FIG. 8

METHOD AND SYSTEM FOR TRANSFORMING INPUT DATA STREAMS

BACKGROUND

The field of the invention relates to data transformation, and more particularly, to apparatus and method for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats.

Businesses communication has become increasingly complex. The demands of business trends such as Customer Relationship Management and Supply Chain Management combined with emerging communication technologies, which allow business partners to share information instantly, are mainly responsible for this communication evolution. The number of business partners and the means with which they collaborate (using e-mail, fax, public internet and mobile devices) are steadily increasing. Adding to this complexity, a growing number of customers and suppliers require that the communication be tailored to their specific needs. In short, businesses today need to provide communication processes that are automated and personalized. Meeting this challenge requires a new understanding of business communications in the age of the Internet. Thus, there is a need for better control of the complexity of business communication.

BRIEF DESCRIPTION OF THE DRAWINGS

The features of the present invention, which are believed to be novel, are set forth with particularity in the appended claims. The invention may best be understood by reference to the following description taken in conjunction with the accompanying drawings, in the several figures of which like reference numerals identify like elements, and in which:

FIG. **1** is a general block diagram of one embodiment of a system for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats;

FIG. **2** is a more detailed block diagram of one embodiment of the system;

FIG. **3** is a further block diagram of an implementation of one embodiment of the system;

FIG. **4** is a block diagram of one embodiment of a portion of the system;

FIG. **5** is a block diagram of another embodiment of a portion of the system;

FIG. **6** is a block diagram of yet another embodiment of a portion of the system;

FIG. **7** is a diagram of run-time phases of an embodiment of the system; and

FIG. **8** is block diagram of a further embodiment of a portion of the system.

DETAILED DESCRIPTION

While the present invention is susceptible of embodiments in various forms, there is shown in the drawings and will hereinafter be descried some exemplary and non-limiting embodiments, with the understanding that the present disclosure is to be considered an exemplification of the invention and is not intended to limit the invention to the specific embodiments illustrated.

One embodiment of a system for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats is depicted in FIG. **1**. A plurality of input connector modules **100**, **102**, **104** receive respective input data streams **106**, **108**, **110**. A plurality of input queues **112**, **114** store the received input data streams **106**, **108**, **110**. A plurality of job threads **116**, **118**, **120**, **122** are operatively connected to respective input queues **112**, **114**. Each job thread (**116**, **118**, **120**, **122**) in parallel with at least one other job thread (**116**, **118**, **120**, **122**) formatting a stored input data stream to produce an output data stream (**124**, **126**, **128**, **130**). A plurality of output queues **132**, **134** respectively store the output data streams **124**, **126**, **128**, **130** from the plurality of job threads **116**, **118**, **120**, **122**. A plurality of output connector modules **136**, **138**, **140** are operatively connected to the output queues **132**, **134**, the output connector modules **136**, **138**, **140** supplying respective output data streams (**124**, **126**, **128**, **130**). It is to be understood that the novel system may have any number of input connector modules **100**, **102**, **104**, input queues **112**, **114**, job threads **116**, **118**, **120**, **122**, output queues **132**, **134**, and output connector modules **136**, **138**, **140**. Also, there is no restriction on how they may be shared and FIG. **1** is only one example of a system configuration. Furthermore, a job thread may be directly connected to an input connector and/or to an output connector (see job thread **122** and output connector **140** in FIG. **1**, for example).

FIG. **2** depicts an embodiment of the system in more detail. An input data stream **200** from a source device **202** or application (provider) is evaluated and manipulated based on the data content, transmission protocol and data format requirements of the receiving device **204** or application (consumer). Input can originate from a number of sources, refined and then multiplexed to multiple output channels. Thus, one-to-many and many-to-many processing from provider to consumer is possible.

The input is processed according to communication rules **224**, which define how the content is transformed, delivered and presented to the consumer. The communication rules **224** are applied based on matching the input from the source device **202** to the requirement of the receiver device **204** of the output data stream **208**.

At runtime, the input data stream **200** is described in an event parsing model **210** and a corresponding transformation model **212** upon which the data transformation is based. The data stream is manipulated based on mapping rules **214** in the transformation model **212**, communication rules **216** in the process model **218** and the content and structure of the input event.

The event parsing model **210**, transformation model **212**, and process model **218** are statically defined in a design phase and determine the global framework for the communication process between the provider (source device **202**) and the consumer (receiving device **204**). The input event parsing model **210** is defined using an event tool **220**, which defines the sequences and patterns to detect in the input data stream **200**. The transformation model **212** can correspond to the event parsing model **210** or can consist of a combination of events derived from the data stream or from additional mapping rules defined at design time in a mapping tool **222**. The processing rules **224** for the presentation and delivery to the output data stream is defined in the process tool **226**.

External communication rules for the processing and delivery of the information personalized for the consumer is derived from a matching consumer model **230** at run time. The consumer model **230** is dynamic and need not be predefined before the information is transformed or pro-

cessed at runtime. The consumer model **230** is applied to the processing model **218** to determine the actual communication rules **206**.

The event tool **220**, the mapping tool **222**, and the process tool **226** occur in the design phase **232**. The event parsing model **210**, the transformation model **212**, and the process model **218** form the provider schema **234**. In the runtime phase **236** the input data stream **200** is received by an event agent **238**, which parses the input data stream **200**. A transformation engine **240** effects the actual transformation of the data from one format to another format. A process engine **242** then applies the communication rules **224** and sends the output data stream **208** to the receiving device **204**.

The multi-threading system increases the performance and provides support for parallel job execution. This system architecture also offers better scalability for multi-processor systems. All threads are connected to queues and/or connectors, enabling extremely flexible configuration. Several job threads can serve one or several queues and several input connectors can use one or several queues and job threads.

In one embodiment job threads pick up data from the queue in the same order as it was stored. Jobs that arrive via input connectors are stored in input queues, and job threads pick up the jobs and execute them independently of other job threads. When an input connector has written a job to a queue, that connector is immediately ready to receive more data; it does not have to wait for the system to process previous jobs. After processing, jobs are stored in output queues, from where output connectors can pick them up and pass them on to their final destination. Thus, the use of queuing is one embodiment of the system.

The following is a more detailed description of the operation of the system and method for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats.

In the embodiment depicted in FIG. **3**, the server **300** is the "main engine" and is configured using a project tool **302**. All configurations are defined in the project tool **302** and then exported in two text files **304**, **306** for platform configuration and message configuration to the server **300**. The server **300** reads these files **304**, **306** at startup and creates and connects events **308**, processes **310** and queues **312**, **314** according to the instructions in the files **304**, **306**. This embodiment focuses on how the server **300** builds its pipelines and how it processes data **316** from a business application **318** and provides output data **320**. The system is applicable to other applications, which need to reformat data streams. During an initiation phase the project tool **302** uses a sample file **322** from the business application **318**. As will be explained below, the server **300** has a parsing model **324** and a runtime model **326**.

The system is multi-threading, but for the purpose of describing the operation of the system, the threading model is considered to consist of a main thread **400** and input threads, such as input thread **402** (see FIG. **4**). The main thread **400** is responsible for initiation. It parses all command line options, all driver files and all export files from the project tool. Based on this information it creates the parsing model **404**. Finally it creates one input thread **402** for each input queue, starts these threads and then becomes passive. It remains passive until it gets a signal that a user wants to terminate the server. When this occurs, it stops all input threads, de-allocates all resources and exits. Each input thread listens to a physical port from which it can receive data and execute any jobs found on this port.

The parsing model **404** is created as a read-only object by the main thread **400** at startup and cannot be changed. The parsing model **404** contains all the information specified by the user in the project tool. This information is exported to the server and stored in the parsing model **404**.

The parsing model **404** communicates with the objects in the runtime model and provides information such as: agent information, which is information about which agent **406** a thread job manager **408** shall use; variable information, which is information about which variables to create and instantiate; message structure, which is information about how to structure a message (such as messages **410**, **412**); output action, which is how the process communicates with the parsing model **404** to receive instructions about which actions to take (These actions may include sending output to the output pipeline **414**, running a script or carrying out sorting, for example); sorting information, which is information about whether sorting should be done or not; output pipeline objects information, which is information regarding how the thread job manager **408** creates the output pipeline **414** and makes sure that the required objects are inserted into the pipeline **414** based on information in the parsing model **404**; events and processes information, which is information regarding which events **416** to detect in the data stream and which processes to launch when an event **416** is detected.

The runtime model contains components that are created at start-up and dynamic components that are created during runtime. The main thread **500** creates the parsing model **502** and all input threads, such as input thread **504**. These components cannot be changed during the session. All other components, events, messages and output pipeline objects, are dynamically created at runtime.

The following is a step-by-step description of an example of the flow in one embodiment of the runtime model.

1. When the server starts, the main thread **500** creates the parsing model and all input threads **504** by using information in the files exported from the project tool. When this is done, the main thread becomes idle and listens only to a server shutdown command. When this occurs, the main thread **500** is responsible for closing all input threads **504**.
2. Input data (from a business application, for example) is received by a physical input **506**.
3. A filter **508** in the input thread **504** ensures that only relevant data is passed to an agent **510**.
4. When the agent **510** receives the data, the collect-phase begins. In this phase the agent **510** reads the entire input file and then carries out the following steps for each event **512** in the job:

4.1. The event **512** is identified and the data is retrieved from it.

4.2. A field list is created for the event **512**.

4.3. The retrieved script for the event **512** is run. Once these steps have been carried out for each event **512**, sorting (if any) is performed using variables set in the events **512** and the retrieved scripts.

5. The collect phase is now complete.
6. When the thread job manager **514** receives permission from the global thread manager, the first event **512** is created by the thread job manager **514**. Information about how to create the event **512** is retrieved from the parsing model **502**.
7. The agent **510** fills the event with fields.
8. The event **512** creates a message **516** based on the event's field list and the information in the parsing

model **502**. A message tree is built using fields, blocks and variables. The message **516** is then passed on to the thread job manager **514**.

9. The thread job manager **514** runs "script before event".
10. The thread job manager **514** creates a process **520** by using information in the parsing model **502** and message **518**.
11. The thread job manager **514** runs "script before process".
12. A check is made to determine if this process should be skipped. A skip can be forced by a rule attached to the process **529** or by executing a script function "skip()" in the "script before process".
13. If no skip is detected, the thread job manager **514** creates the output pipeline **522** for the process **520**. This is based on the information in the parsing model **504**. The process **520** is then executed according to the instructions in the parsing model **504** and in the data flow. The output pipeline **522** may contain objects, such as sort/archive **524**, driver **526**, physical output **528**. The output pipeline **522** may be operatively connected to a receiving device **530**.
14. When the process **520** is finished, "script after process" is executed.
15. Steps **12** to **14** are repeated for all processes **520** defined for the event **512**.
16. When all processes **520** are created the thread job manager **514** runs "script after event".
17. Steps **9** to **16** are performed for each event **512**.

In another embodiment depicted in FIG. **6** an input pipeline (input thread **600**) consists of a pipeline of objects that are connected through one data channel and one message channel. The pipeline **600** always starts with a physical input object **602** and ends with a thread job manager **604**. Other objects can be inserted between the physical input object **602** and the thread job manager **604**. These objects can perform various operations with the data as long as they send it to the next object in the pipeline. Normally these objects are filters **606** that remove unwanted data from the data channel.

Each input thread **600** consists of only one input pipeline. Its only task is to find incoming jobs arriving at the physical input object **602** and send jobs down to the different objects in the pipeline. Eventually, it reaches the thread job manager **604** that processes a job.

The physical input object **602** is a physical port through which incoming data is received. It is also the start of the input thread data pipeline. A physical port may be one of the following types: serial (receives data directly from a serial port); directory scan (scans a file system directory for files that match a file search criterion); device (listens directly to a hardware device, e.g. a parallel port); standard input (listens to standard input); TCP/IP sockets (listens to a socket for incoming data); named pipe; (listens to a named pipe); internal (data is sent from a server output queue in the same system); netware bindery (acts as a NetWare printer); netware NDS (acts as a NetWare NDS printer).

The physical input object **602** starts to listen for incoming data. As soon as the physical input object **602** detects an incoming job the physical input object **602** sends the job down the input thread data pipeline byte by byte as raw data. How ports are listened to depend on the type of port.

If a filter has been chosen for the input queue in project tool, an input filter object **606** is inserted in the input thread data pipeline **600** after the physical input object **602**. If several filters have been chosen, several filter objects are inserted in serial in the pipeline **600**.

A filter's task is to remove unwanted sequences or to convert sequences in the incoming data stream. An example of removing sequences is a filter that removes PCL escape codes and just sends the actual PCL document data to the next object in the pipeline. An example of converting is a filter that receives compressed (zipped) data and uncompresses (unzips) it before sending it to the next object.

The script language makes it possible at runtime to decide what output to produce and to which queue to send it. The script language is an event driven procedural language.

The input thread data pipeline of the input thread **600** always ends with a thread job manager **604**. Each thread job manager **604** contains an agent **610**. The thread job manager **604** is responsible for detecting events and launching and controlling the events and processes.

An agent **610** is the interface between the thread job manager **604** and the input thread data pipeline and receives the incoming data. It is responsible for detecting events and extracting fields in the raw data input stream. There may be several different agents **610**; each specialized for a specific type of input. For example, one agent for record based input from mainframes, another agent for XML data. The agent to use is specified in the project tool. The thread job manager **604** finds this information in the parsing model **612**. In one embodiment the agent **610** receives data as one page and breaks it down into a field list.

The agent **610**, when a job arrives and when events are found in the job, notifies the thread job manager **604**. The thread job manager's main task is to control the execution of the job (i.e. the events, scripts, sorting and processes of the job). When executing the job, the thread job manager **604** creates events and processes and makes sure that they are executed in the right order. When processes are executed, the thread job manager **604** is also responsible for setting up the output pipeline **616** for the process.

In general, the main task for the process is to produce output and send it to an output pipeline. The data may be received as a message containing blocks that contain fields. In this embodiment the execution is block driven, meaning that the process identifies all blocks in the message and then communicates with the parsing model to get instructions about which actions to take for each block, for example, to send output to the output pipeline, to run a script or to perform sorting. The type of output created differs depending on the type of process used.

The following are examples of types of processes. The process "PageOUT" produces a page layout. This is by far the most complicated process and is used for creating documents for printing, faxing, PDF, web etc. The process "StreamOUT" produces flat field and record based text files. The process "XMLOUT" produces XML output. This is a special version of "StreamOUT". The process "MailOUT" produces e-mail and can also attach the result of another process to the e-mail. The process "SMSOUT" produces SMS messages that can be sent to mobile phones.

In another embodiment output sent to the output pipeline is sent as meta records containing instructions for the device drivers. An example of a meta record is as follows: output the text ", Inc." at position x=346 and y=345 using font Arial size 10. When fields and variables are used in the output, the process retrieves the current field or variable value. This means that a reference to a field or variable is never included in meta records. Instead, the value of the field or variable is sent. To the output pipeline objects, it is transparent if it is static text or text from the incoming data that is being

delivered. The device drivers convert Meta records to device specific output. The device drivers are part of the output pipeline.

In thread job execution the thread job manager splits all requests that receive and process into jobs. Each job consists of one or more events together with all processes belonging to these events. The processes can send their output to one or more output pipelines. Each of these pipelines produce one output entity for the complete job. For example if 30 invoices are received at the input pipeline and a "PageOUT" process produces 30 invoices and sends the invoices to a spooler system, these 30 invoices being sent as one print job to the spooler.

The default scope of a job is that each input file will result in one job. However, the incoming file may be split the incoming file into several smaller jobs. The smallest possible job is when the job consists of only one event. The thread job manager (actually the thread job manager agent) is responsible for deciding when a job starts and ends. Normally this is straight forward since one incoming request to a physical input object will result in one job.

There can be many reasons for dividing a large job into smaller jobs. For example, there may be one entry in the spooler system for each process, for example for each invoice. In a further embodiment some settings may be sent to the output queue. This is usually performed at the beginning of the job, for example downloading overlay files to a printer.

One example of an implementation of the system occurs when an external application that is required to process an output job sends this job as one file to the system. When the agent receives the job and recognizes it as something that should trigger an event, the job begins. This sends signals to the thread job manager for the job to begin and for the collect phase **700** to begin (see FIG. **7**).

The agent will now start to scan the input for fields and new events. All fields found are stored in a list that is associated with the current event. If, in the parsing model, the field is designated to create a variable, this is done at this stage. If a new event is found it will be added to a list of found events, and any fields found after this will be associated with this event. This process continues until a list of all events, with all fields, has been created. This signals an end of the collect phase **700** to the thread job manager. The Collect phase is necessary for creating this list, which in turn is used to sort the incoming events. Information is stored in the parsing model about whether or not sorting should be carried out.

The thread job manager will now pre-process all events and processes belonging to the job in a pre-process phase **702**. During the pre-process phase **702** the whole job is executed, but without sending anything to the output pipeline. The pre-process phase **702** is used, for example, to calculate the number of pages and where page breaks occur and to determine which resources are to be used. A resource may, for example, be an overlay that should be sent to a printer. It is also possible to cancel the job, that is undo everything that has been done in the job and skip the rest of the input. This can be done conditionally, based on input field values, in scripts. Event and process execution is carried out in the pre-process phase **702** in the following order:

1. The first event in the event list is pre-processed first, then all the processes for this event.
2. The next event in the event list, together with its processes, is preprocessed.
3. This continues until all the events in the list have been pre-processed.

Note that this is the order after events have been sorted. Before and after each event and process a script is run. In this script, the process can conditionally be skipped.

Now the thread job manager has stored all information needed from the pre-process phase **702** and can execute the events and processes in a process phase **704**. First, it performs a rollback on everything. For example, variables are restored to their values before the pre-process phase **702** and ODBC operations that have been executed in a transaction are rolled-back. Next it sends any resources (for example, overlays) that were found during the pre-process phase **702** to the output pipeline. The events and processes are executed in the process phase **704** in the same order as in the pre-process phase **702**. The difference is that this time the output is actually sent to the output pipeline. After the last process is executed, the job is complete. The thread job manager releases all resources that were temporarily assigned.

In FIG. **8** the output pipeline **800** consists of a pipeline of objects that are connected through one data channel and one message channel. The pipeline **800** always starts with a process **802** and ends with a physical output object **804**. Between the process **802** and the physical output object **804** other objects may be inserted. These objects may be used to perform various operations with the data and then pass the data on to the next object in the pipeline **800**. Examples of operations that may be performed in various embodiments are sorting, or splitting the pipeline into two branches (such as sorting object **806**). Also one of the objects may be a device driver **808** that converts the meta data into formatted data.

The physical output object **804** always points to a physical destination, such as receiving device **810**. This can, for example, be a printer or an e-mail server. The physical output object **804** is responsible for the actual delivery of the output data to its final destination.

Different objects may be included in the pipeline **800** depending on information in the parsing model. The thread job manager creates the output pipeline **800** and ensures that the required objects are inserted in the pipeline **800**. The thread job manager also connects the pipeline **800** to the process **800**.

In one embodiment the following rules may apply to all output pipelines in the system: Each physical output object corresponds to one, and only one, queue as defined in the parsing model. There may only be one pipeline for each physical output object. The physical output object for a pipeline is always the same throughout an entire job. The pipeline is always connected to one process at a time. These rules imply that output from different processes in the same job, that use the same physical output object, will be kept together, that is, delivered as one unit to the final destination, for example a spooler system.

In the data channel, the process sends meta records down the pipeline. If there is a device driver in the pipeline, it reformats the meta record according to the format expected by the destination. Eventually the information reaches the physical output object, which sends it to a physical destination, for example, a spooler system or a file. The message channel is used by the thread job manager to send messages to notify the objects in the pipeline when certain events occur.

Output processors or objects may be inserted anywhere in the pipeline. These processors may change the data that is sent through the data channel. It is also possible to use a

pipeline without any output processors, that is a pipeline with just a device driver and a physical output object.

Thus in general terms the present system (and the corresponding method) is for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats. A plurality of input connector modules receive respective input data streams and at least one input queue stores the received input data streams. A plurality of job threads is operatively connected to the at least one input queue, each job thread, in parallel with at least one other job thread, formatting a stored input data stream to produce an output data stream. At least one output queue respectively stores the output data streams from the plurality of job threads. A plurality of output connector modules is operatively connected to the at least one output queue, the output connector modules supplying respective output data streams.

In an embodiment each of the job threads has at least one event agent associated with at least one parsing model, the event agent having an input port that receives an input data stream, and having an output port. At least one transformation engine is associated with at least one transformation model, the transformation engine having an input port operatively connected to the output port of the event agent. At least one process engine is associated with at least one process model, the process engine having an input port operatively connected to the output port of the transformation engine, and having an output port for supplying an output data stream. The transformation model has mapping rules for manipulating the input data stream, and the process model has communication rules for formatting the output data stream.

In another embodiment the at least one input queue may be shared between the input connector modules and the job threads, and the at least one output queue may be shared between the job threads and the output connectors. The job threads may receive input data streams in the order in which the input data streams are stored in the at least one input queue. In general, the job threads receive input data streams from the at least one input queue, format the input data streams into output data streams, and store the output data streams in the at least one output queue, independent of one another and in parallel.

It is to be understood, of course, that the present invention in various embodiments can be implemented in hardware, software, or in combinations of hardware and software.

The present invention is not limited to the particular details of the apparatus and method depicted, and other modifications and applications are contemplated. Certain other changes may be made in the above-described apparatus and method without departing from the true spirit and scope of the invention herein involved. It is intended, therefore, that the subject matter in the above depiction shall be interpreted as illustrative and not illuminating sense.

What is claimed is:

1. A system for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats, comprising:

a parsing model comprising information about message structure;

at least one input queue for storing input data streams;

a plurality of job threads operatively connected to the at least one input queue, each job thread, in parallel with at least one other job thread, formatting a respective stored input data stream to produce an output data stream, each job thread comprising:

a filter for filtering irrelevant data from the respective retrieved input data stream;

an event identifier for identifying events in the input data stream, the events comprising sequences, patterns, or both, in the input data stream;

a field list for each of the identified events, the fields comprising data, variables, or both in the input stream;

a message generator for generating messages associated with each of said identified event in response to the field list for said identified event and said information about message structure; and

a processor for processing each message in response to said information about message structure to generate meta records, the meta records comprising data that is not formatted for a specific output device; and

a formatter for formatting the meta records to produce an output data stream;

at least one output queue for respectively storing the output data streams from the plurality of job threads.

2. The system according to claim **1**, wherein the input queues are shared between input connector modules and the job threads, and wherein the output queues are shared between the job threads and output connectors.

3. The system according to claim **1**, wherein the job threads receive input data streams in the order in which the input data streams are stored in the input queues.

4. The system according to claim **1**, wherein the job threads, independent of one another and in parallel, receive input data streams from the input queues, format the input data steams into output data steams, and store the output data streams in the output queues.

5. A system for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats, comprising:

a parsing model comprising information about message structure;

a plurality of input connector modules for receiving respective input data streams;

a plurality of input queues for storing received input data streams;

a plurality of job threads operatively connected to at least one of respective input connector modules of the plurality of input connector modules and respective input queues of the plurality of the input queues, each job thread in parallel with at least one other job thread formatting a stored input data stream to produce an output data stream, each job thread comprising:

filter for filtering irrelevant data from the respective retrieved input data stream;

an event identifier for identifying events in the input data stream, the events comprising sequences, patterns, or both, in the input data stream;

a field list for each of the identified events, the fields comprising data, variables, or both in the input stream;

a message generator for generating messages associated with each of said identified event in response to the field list for said identified event and said information about message structure; and

a processor for processing each message in response to information about message structure to generate meta records, the meta records comprising data that is not formatted for a specific output device; and

a formatting for formatting the meta records to produce an output data stream;

a plurality of output queues for respectively storing the output data streams from the plurality of job threads; and

a plurality of output connector modules operatively connected to at least one of the output queues and the job threads, the output connector modules supplying respective output data streams.

6. The system according to claim **5**, wherein each of the job threads comprises:

at least one event agent associated with at least one parsing model, the event agent having an input port that receives an input data stream, and having an output port;

at least one transformation engine associated with at least one transformation model, the transformation engine having an input port operatively connected to the output port of the event agent; and

at least one process engine associated with at least one process model, the process engine having an input port operatively connected to the output port of the transformation engine, having an output port for supplying an output data stream.

7. The system according to claim **6**, wherein the transformation engine has mapping rules for manipulating the input data stream.

8. The system according to claim **6**, wherein the process engine has communication rules for formatting the output data stream.

9. The system according to claim **5**, wherein the input queues are shared between the input connector modules and the job threads, and wherein the output queues are shared between the job threads and the output connectors.

10. The system according to claim **5**, wherein the job threads receive input data streams in the order in which the input data streams are stored in the input queues.

11. The system according to claim **5**, wherein the job threads, independent of one another and in parallel, receive input data streams from the input queues, format the input data streams into output data streams, and store the output data streams in the output queue.

12. A system for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats, comprising:

a parsing model comprising information about message structure;

a plurality of input connector modules for receiving respective input data streams;

at least one input queues for storing received input data streams;

a plurality of job threads operatively connected to the at least one input queue, each job thread, in parallel with at least one other job thread, formatting a stored input data stream to produce an output data stream, each job thread comprising:

a filter for filtering irrelevant data from the respective retrieved input data stream;

an event identifier for identifying events in the input data stream, the events comprising sequences, patterns, or both, in the input data stream;

a field list for each of the identified events, the fields comprising data, variables, or both in the input stream;

a message generator for generating messages associated with each of said identified event in response to the field list for said identified event and said information about message structure; and

a processor for processing each message in response to said information about message structure to generate meta records, the meta records comprising data that is not formatted for a specific output device; and

a formatter for formatting the meta records to produce an output data stream;

at least one output queue for respectively storing the output data streams from the plurality of job threads; and

a plurality of output connector modules operatively connected to the at least one output queue, the output connector modules supplying respective output data streams.

13. The system according to claim **12**, wherein each of the job threads comprises:

at least one event agent associated with at least one parsing model, the event agent having an input port that receives an input data stream, and having an output port;

at least one transformation engine associated with at least one transformation model, the transformation engine having an input part operatively connected to the output port of the event agent; and

at least one process engine associated with at least one process model, the process engine having an input port operatively connected to the output port of the transformation engine, having an output port for supplying an output data stream.

14. The system according to claim **13**, wherein the transformation model has mapping rules for manipulating the input data stream.

15. The system according to claim **13**, wherein the process model has communication rules for formatting the output data stream.

16. The system according to claim **12**, wherein the at least one input queue is shred between the input connector modules and the job threads, and wherein the at least one output queue is shared between the job threads and the output connectors.

17. The system according to claim **12**, wherein the job threads receive input data streams in the order in which the input data streams are stored in the at least one input queue.

18. The system according to claim **12**, wherein the job threads, independent of one another and in parallel, receive input data streams from the at least one input queue, format the input data streams into output data streams, and store the output data streams in the at least one output queue.

19. A method for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats, comprising:

providing a parsing model comprising information about message structure;

storing input data streams in at least one input queue;

retrieving input data streams from the at least one input queue;

formatting each respective retrieved input data stream by a respective job thread, the respective job thread formatting the respective retrieved input data stream in parallel with at least one other job thread that formats another respective retrieved input data stream, each job thread:

filtering irrelevant data from the respective retrieved input data stream;

identifying events in the input data stream, the events comprising sequences, patterns or both, in the input data stream;

creating a field list for each of the identified events, the fields comprising data, variables, or both in the input stream; and

generating message associated with each of said identified event in response to the field list for said identified event and said information about message structure;

processing each message in response to said information about message structure to generate meta records, the meta records comprising data that is not formatted for a specific output device;

formatting the meta records into an output data stream; and

storing the respective output data streams in at least one output queue.

20. The method according to claim **19**, wherein the at least one input queue is shared between the input connector modifies and the job threads, and wherein the at least one output queue is shared between the job threads and the output connectors.

21. The method according to claim **19**, wherein the job threads receive input data streams in an order in which the input data streams are stored in the at least one input queue.

22. A system for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats, comprising:

a parsing model comprising information about message structure;

means for storing input data streams in at least one input queue;

means for retrieving, via job threads, input data streams from the at least one input queue;

means for formatting a respective retrieved input data stream by a respective job thread to produce a respective output data stream, the respective job thread formatting the respective retrieved input data stream in parallel with at least one other job thread that formats another respective retrieved input data stream to produce another respective output data stream, each job thread:

filtering irrelevant data from the respective retrieved input data stream;

identifying events in the input data stream, the events comprising sequences, patterns or both, in the input data stream;

creating a field list for each of the identified events, the fields comprising data, variables, or both in the input stream; and

generating message associated with each of said identified event in response to the field list for said identified event and said information about message structure;

means for processing each message in response to said information about message structure to generate meta records, the meta records comprising data that is not formatted for a specific output device;

means for formatting the meta records into an output data stream; and

means for storing the respective output data streams in at least one output queue.

23. A method for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats, comprising:

providing a parsing model comprising information about message structure;

receiving input data streams, each of the input data streams being in a respective data format of a plurality of first data formats;

storing the input data streams in at least one input queue;

retrieving, via job threads, stored input data streams from the at least one input queue;

formatting a respective retrieved input data stream that is in a first data format of a plurality of first data formats by a respective job thread, the formatting by the respective job thread of the respective retrieved input data stream being in parallel with formatting by at least one other job thread of another respective retrieved input data stream, each job thread:

filtering irrelevant data from the respective retrieved input data stream;

identifying events in the input data stream, the events comprising sequences, patterns or both, in the input data stream;

creating a field list for each of the identified events, the fields comprising data, variables, or both in the input stream; and

generating message associated with each of said identified event in response to the field list for said identified event and said information about message structure;

processing each message in response to said information about message structure to generate meta records, the meta records comprising data that is not formatted for a specific output device;

formatting the meta records into an output data stream; and

storing the output data streams in at least one output queue;

selecting one of the output data streams in the at least one output queue; and

outputting the selected output data stream.

24. The method according to claim **23**, wherein the at least one input queue is shared between input connector modules and the job threads, and wherein the at least one output queue is shared between the job threads and output connectors.

25. The method according to claim **23**, wherein the job threads receive input data streams in an order in which the input data streams are stored in the at least one input queue.

26. A computer readable storage medium containing embedded computer program code for transforming an input data stream in a first data format of a plurality of first data formats to an output data stream in a second data format of a plurality of second data formats, the computer readable storage media containing computer program code segments comprising:

a first computer program code segment that provides a parsing model comprising information about message structure;

a second computer program code segment that stores input data streams in at least one input queue;

a third computer program code segment that retrieves, via job threads, input data streams from the at least one input queue; and

a fourth computer program code segment that formats a respective retrieved input data stream by a respective job thread to produce a respective output data stream, the respective retrieved input data stream being for-

matted in parallel with at least one other respective retrieved input data stream, each job thread:

filtering irrelevant data from the respective retrieved input data stream;

identifying events in the input data stream, the events comprising sequences, patterns or both, in the input data stream;

creating a field list for each of the identified events, the fields comprising data, variables, or both in the input stream; and

generating message associated with each of said identified event in response to the field list for said identified event and said information about message structure;

a fifth computer program code segment that processes each message in response to said information about message structure to generate meta records, the meta records comprising data that is not formatted for a specific output device;

a sixth computer program code segment that formats the meta records into an output data stream; and

a seventh computer program code segment that stores the output data streams in at least one output queue.

27. The computer readable storage medium according to claim **26**, wherein the computer readable storage medium further comprises for forming a job thread;

an eighth computer program code segment that forms at least one event agent associated with at least one parsing model, the event agent receiving an input data stream;

a ninth computer program code segment that forms at least one transformation engine associated with at least one transformation model, the transformation engine receiving the input data stream from the event agent; and

a tenth computer program code segment that forms at least one process engine associated with at least one process model, the process engine receiving the transformed input data stream and supplying an output data stream.

* * * * *